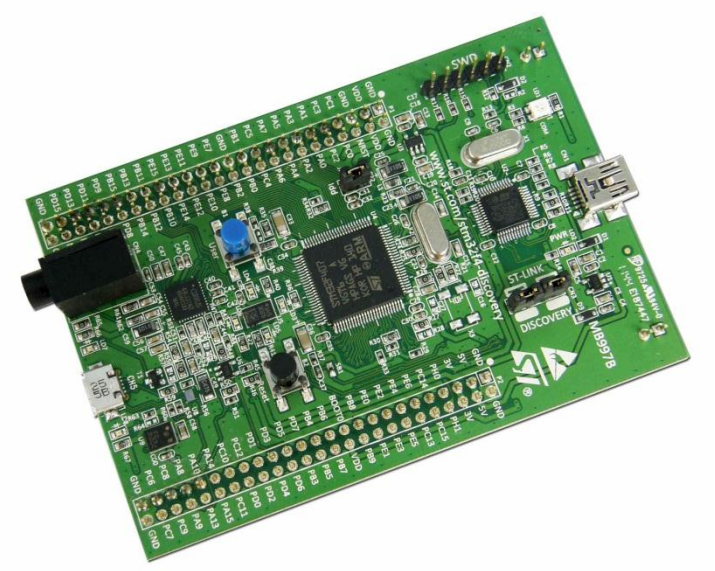
Politechnika Poznańska, Instytut Automatyki i Inżynierii Informatycznej

Podstawy technik mikroprocesorowych

Ćwiczenia laboratoryjne 9 – DMA



Michał Fularz
2012-12-04

1. Wstęp teoretyczny.

Na zajęcia obowiązuje znajomość materiału zaprezentowanego na wykładach.
W układach z rodziny STM32F4 znajdują się dwa niezależne układy DMA, każdy z nich umożliwia utworzenia 8 połączeń (*Stream*), z czego transfer dla danego połączenia może wywoływać jeden z 8 kanałów (*Channel*). Wszystkie możliwości zaprezentowano w tabelach poniżej (dostępne w Reference Manual dla STM32F4).

Table 22. DMA1 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX		SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX		SPI3_TX
Channel 1	I2C1_RX		TIM7_UP		TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1		I2S3_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5			TIM3_CH4 TIM3_UP		TIM3_CH1 TIM3_TRIG	TIM3_CH2		TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2		TIM5_UP	
Channel 7		TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

Table 23. DMA2 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1		TIM8_CH1 TIM8_CH2 TIM8_CH3		ADC1		TIM1_CH1 TIM1_CH2 TIM1_CH3	
Channel 1		DCMI	ADC2	ADC2				DCMI
Channel 2	ADC3	ADC3				CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX		SPI1_RX	SPI1_TX		SPI1_TX		
Channel 4			USART1_RX	SDIO		USART1_RX	SDIO	USART1_TX
Channel 5		USART6_RX	USART6_RX				USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
Channel 7		TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3			TIM8_CH4 TIM8_TRIG TIM8_COM

Przebieg ćwiczenia.

a. Tworzenie projektu.

Utworzyć nowy projekt (lub skorzystać z programu z poprzednich zajęć). W zakładce ***Repositories*** dodać następujące moduły:

- DMA – funkcje obsługi modułu bezpośredniego dostępu do pamięci (*ang. **D**irect **M**emory **A**ccess*), dodaje pliki stm32fxx_dma.h oraz stm32fxx_dma.c do projektu.

W oknie edytora kodu otworzyć załączone pliki (aby je podświetlić w oknie ***Components*** wybrać odpowiedni moduł (MISC, USART)). Zwrócić uwagę na:

- Exported types (stm32fxx_dma.h – DMA_InitTypeDef, opis poszczególnych pól w pliku oraz na wykładzie),
- Exported functions (stm32fxx_dma.h – DMA_Init, DMA_Cmd, DMA_GetFlagStatus, DMA_ClearFlag).

W pliku stm32f4xx_dma.c znajduje się krótki opis funkcjonalności modułu (na początku pliku, w komentarzu), a także definicje funkcji wraz z ich krótki opisem (indeksowanym przez IDE i wyświetlanym po umieszczeniu kursora na funkcji i naciśnięciu klawisza F2).

Do pliku z funkcją **main** programu dodać odpowiednie biblioteki (#include ”stm32f4xx_dma.h” oraz #include ”misc.h”).

b. Transfer danych z układu peryferyjnego do pamięci (peripheral to memory).

W celu przesyłania danych za pomocą DMA z bloku peryferyjnego do pamięci bez udziału procesora wykorzystany zostanie moduł ADC. Implementacja przedstawiona w ćwiczeniu dotyczącym ADC zakładała wyzwolenie programowe przetwornika, następnie oczekiwanie na zakończenie konwersji, a na koniec odczytanie wyniku z rejestrów układu ADC. Rozwiązanie to znacząco obciążało procesor. Za prezentowana poniżej implementacja z wykorzystaniem DMA dokonuje konwersji bez przerwy, a wynik jest przesyłany do pamięci bez udziału procesora.

W celu przygotowania DMA do działania należy (inicjalizację zamieszczono w jednej funkcji):

```
void MY_DMA_initP2M(void)
{
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);

	DMA_InitTypeDef strukturaDoInicjalizacjiDMA;

	// wybór kanału DMA
```

```
	strukturaDoInicjalizacjiDMA.DMA_Channel = DMA_Channel_0;
	// ustalenie rodzaju transferu (memory2memory / peripheral2memory /
memory2peripheral)
	strukturaDoInicjalizacjiDMA.DMA_DIR = DMA_DIR_PeripheralToMemory;
	// tryb pracy - pojedynczy transfer bądź powtarzany
	strukturaDoInicjalizacjiDMA.DMA_Mode = DMA_Mode_Circular;
	// ustalenie priorytetu danego kanału DMA
	strukturaDoInicjalizacjiDMA.DMA_Priority = DMA_Priority_Medium;
	// liczba danych do przesłania
	strukturaDoInicjalizacjiDMA.DMA_BufferSize = (uint32_t)1;
	// adres źródłowy
	strukturaDoInicjalizacjiDMA.DMA_PeripheralBaseAddr =
(uint32_t)(ADC_1_ADDRESS_BASE+ADC_DR_ADDRESS_OFFSET);
	// adres docelowy
	strukturaDoInicjalizacjiDMA.DMA_Memory0BaseAddr = (uint32_t)&valueFromADC;
	// okreslenie, czy adresy mają być inkrementowane po każdej przesłanej
paczce danych
	strukturaDoInicjalizacjiDMA.DMA_MemoryInc = DMA_MemoryInc_Disable;
	strukturaDoInicjalizacjiDMA.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
	// ustalenie rozmiaru przesyłanych danych
	strukturaDoInicjalizacjiDMA.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_HalfWord;
	strukturaDoInicjalizacjiDMA.DMA_MemoryDataSize =
DMA_MemoryDataSize_HalfWord;
	// ustalenie trybu pracy - jednorazwe przesłanie danych
	strukturaDoInicjalizacjiDMA.DMA_PeripheralBurst =
DMA_PeripheralBurst_Single;
	strukturaDoInicjalizacjiDMA.DMA_MemoryBurst = DMA_MemoryBurst_Single;
	// wyłączenie kolejki FIFO (nie używana w tym przykadzie)
	strukturaDoInicjalizacjiDMA.DMA_FIFOMode = DMA_FIFOMode_Disable;
	// wypełnianie wszystkich pól struktury jest niezbędne w celu poprawnego
działania, wpisanie jednej z dozwolonych wartosci
	strukturaDoInicjalizacjiDMA.DMA_FIFOThreshold = DMA_FIFOThreshold_Full;

	// zapisanie wypełnionej struktury do rejestrów wybranego połączenia DMA
	DMA_Init(DMA2_Stream4, &strukturaDoInicjalizacjiDMA);

	// uruchomienie odpowiedniego połączenia DMA
	DMA_Cmd(DMA2_Stream4, ENABLE);

}
```

Funkcja ta odwołuje się do kilku definicji i adresów (umieścić przed funkcją):

```
#define ADC_1_ADDRESS_BASE                    0x40012000
// ADC_DR = ADC regular Data Register
#define ADC_DR_ADDRESS_OFFSET                 0x4C

volatile uint16_t valueFromADC;
```

Pierwsza definicja określa bazowy adres przetwornika numer 1 (wszystkie rejestry układy peryferyjne są mapowane za pomocą odpowiednich adresów, w celu odczytania adresu dla danego układu należy spojrzeć do noty Reference Manual dla STM32F4, rodział 2.3). Druga definicja podaje offset (przesunięcie) w jakim umieszczony jest rejestr przechowujący wynik konwersji. Suma tych dwóch adresów daje bezwzględny adres danego rejestru (i ten adres jest wpisywany jako

adres źródłowy przy konfiguracji DMA). Globalna zmienna *valueFromADC* rezerwuje obszar pamięci, do którego zapisywana będzie wartość przesłana przez DMA (adres docelowy).

Po skonfigurowaniu DMA należ dokonać konfiguracji modułu ADC:

```
void MY_ADC_init(void)
{
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA , ENABLE);	// wejscie ADC

	//inicjalizacja wejścia ADC
	GPIO_InitTypeDef GPIO_InitStructure;
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init(GPIOA, &GPIO_InitStructure);

	ADC_CommonInitTypeDef ADC_CommonInitStructure;
	// niezależny tryb pracy przetworników
	ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
	// zegar główny podzielony przez 2
	ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;
	// opcja istotna tylko dla tryby multi ADC
	ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
	// czas przerwy pomiędzy kolejnymi konwersjami
	ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
	ADC_CommonInit(&ADC_CommonInitStructure);

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);	//ADC

	ADC_InitTypeDef ADC_InitStructure;
	//ustawienie rozdzielczości przetwornika na maksymalną (12 bitów)
	ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
	//wyłączenie trybu skanowania (odczytywać będziemy jedno wejście ADC
	//w trybie skanowania automatycznie wykonywana jest konwersja na wielu
//wejściach/kanałach)
	ADC_InitStructure.ADC_ScanConvMode = DISABLE;
	//włączenie ciągłego trybu pracy
	ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
	//wyłączenie zewnętrznego wyzwalania
	//konwersja może być wyzwalana timerem, stanem wejścia itd. (szczegóły w
//dokumentacji)
	ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_CC1;
	ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
	//wartość binarna wyniku będzie podawana z wyrównaniem do prawej
	//funkcja do odczytu stanu przetwornika ADC zwraca wartość 16-bitową
	//dla przykładu, wartość 0xFF wyrównana w prawo to 0x00FF, w lewo 0x0FF0
	ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
	//liczba konwersji równa 1, bo 1 kanał
	ADC_InitStructure.ADC_NbrOfConversion = 1;
	// zapisz wypełnioną strukturę do rejestrów przetwornika numer 1
	ADC_Init(ADC1, &ADC_InitStructure);

	// konfiguracja czasu próbkowania sygnału
	ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_84Cycles);

	// włączenie wyzwalania DMA po każdym zakończeniu konwersji
	ADC_DMARequestAfterLastTransferCmd(ADC1, ENABLE);
```

```
	// włączenie DMA dla ADC
	ADC_DMACmd(ADC1, ENABLE);

	// uruchomienie moduły ADC
	ADC_Cmd(ADC1, ENABLE);
}
```

Proszę zwrócić uwagę na dodatkowe funkcje wywoływane na koniec (z przedrostkiem ADC_DMA).

Zadanie:

Skopiować przedstawione fragmenty kodu, a następnie wywołać funkcje (przed główną pętlą) **`MY_DMA_initP2M, MY_ADC_init`** oraz **`ADC_SoftwareStartConv(ADC1);`** (uruchomienie wykonywania konwersji z wartości analogowej na cyfrową, wystarczy wywołać jeden raz, gdyż ADC jest skonfigurowane do pracy ciągłej – patrz konfiguracja ADC, pole `ADC_ContinuousConvMode`). W pętli głównej (**nieskończonej!!!**), w trybie debugowania obserwować wartość odczytaną z przetwornika ADC (`valueFromADC`) podczas zmiany położenia potencjometru. Dopisać zaświecanie diod dostępnych na głównej płytce w zależności od wartości odczytanego napięcia.

c. Transfer danych z pamięci do układu peryferyjnego (memory to peripheral).

W celu zaprezentowania działania DMA w trybie przesyłania wartości z pamięci do układu peryferyjnego wykorzystany zostanie przetwornik DAC. System, będzie w jednakowych odstępach czasu (generowanych przez Timer 6) odczytywał wartość z pamięci i bez udziału procesora (bez ciągłego wywoływania komendy `DAC_SetChannel1Data(DAC_Align_12b_R, value);`) wysyłał do przetwornika DAC.

W celu przygotowania DMA do działania należy (inicjalizację zamieszczono w jednej funkcji):

```
void MY_DMA_initM2P(void)
{
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);

	DMA_InitTypeDef strukturaDoInicjalizacjiDMA;

	// wybór kanału DMA
	strukturaDoInicjalizacjiDMA.DMA_Channel = DMA_Channel_7;
	// ustalenie rodzaju transferu (memory2memory / peripheral2memory /
memory2peripheral)
	strukturaDoInicjalizacjiDMA.DMA_DIR = DMA_DIR_MemoryToPeripheral;
	// tryb pracy - pojedynczy transfer bądź powtarzany
	strukturaDoInicjalizacjiDMA.DMA_Mode = DMA_Mode_Circular;
	// ustalenie priorytetu danego kanału DMA
	strukturaDoInicjalizacjiDMA.DMA_Priority = DMA_Priority_Medium;
```

```
	// liczba danych do przesłania
	strukturaDoInicjalizacjiDMA.DMA_BufferSize = (uint32_t)DMA_DAC_SIGNAL_SIZE;

	// adres źródłowy
	strukturaDoInicjalizacjiDMA.DMA_Memory0BaseAddr =
(uint32_t)DAC_DMA_sine12bit;
	// adres docelowy
	strukturaDoInicjalizacjiDMA.DMA_PeripheralBaseAddr =
(uint32_t)(DAC_CHANNEL_1_ADDRESS_BASE + DAC_DHR12R1_ADDRESS_OFFSET);


	// zezwolenie na inkrementacje adresu po każdej przesłanej paczce danych
	strukturaDoInicjalizacjiDMA.DMA_MemoryInc = DMA_MemoryInc_Enable;
	strukturaDoInicjalizacjiDMA.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
	// ustalenie rozmiaru przesyłanych danych
	strukturaDoInicjalizacjiDMA.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_HalfWord;
	strukturaDoInicjalizacjiDMA.DMA_MemoryDataSize =
DMA_MemoryDataSize_HalfWord;
	// ustalenie trybu pracy - jednorazwe przesłanie danych
	strukturaDoInicjalizacjiDMA.DMA_PeripheralBurst =
DMA_PeripheralBurst_Single;
	strukturaDoInicjalizacjiDMA.DMA_MemoryBurst = DMA_MemoryBurst_Single;
	// wyłączenie kolejki FIFO (nie używana w tym przykadzie)
	strukturaDoInicjalizacjiDMA.DMA_FIFOMode = DMA_FIFOMode_Disable;
	// wypełnianie wszystkich pól struktury jest niezbędne w celu poprawnego
działania, wpisanie jednej z dozwolonych wartosci
	strukturaDoInicjalizacjiDMA.DMA_FIFOThreshold = DMA_FIFOThreshold_Full;

	// zapisanie wypełnionej struktury do rejestrów wybranego połączenia DMA
	DMA_Init(DMA1_Stream5, &strukturaDoInicjalizacjiDMA);

	// uruchomienie odpowiedniego połączenia DMA
	DMA_Cmd(DMA1_Stream5, ENABLE);

	// uruchomienie DMA dla pierwszego kanału DAC
	DAC_DMACmd(DAC_Channel_1, ENABLE);

}
```

Zwrócić uwagę na zmianę adresów – przeanalizować co tym razem jest źródłem, a co docelowym adresem.

Funkcja ta odwołuje się do kilku definicji i adresów (umieścić przed funkcją):

```
#define DAC_CHANNEL_1_ADDRESS_BASE           0x40007400
// DAC_DHR12R1 = DAC Data Holding Register 12 bits, Right aligned channel 1
#define DAC_DHR12R1_ADDRESS_OFFSET           0x08

#define DMA_DAC_SIGNAL_SIZE                         32
const uint16_t DAC_DMA_sine12bit[DMA_DAC_SIGNAL_SIZE] = {
                     2047, 2447, 2831, 3185, 3498, 3750, 3939, 4056, 4095, 4056,
                     3939, 3750, 3495, 3185, 2831, 2447, 2047, 1647, 1263, 909,
                     599, 344, 155, 38, 0, 38, 155, 344, 599, 909, 1263, 1647
                         };
```

Pierwsze dwie definicje – adres układu DAC i offset rejestru przechowującego aktualną konwertowaną wartość ustalono analogicznie do wcześniejszego punktu.

Definicja `DMA_DAC_SIGNAL_SIZE` określa ile elementów będzie miała tablica przechowująca sygnał przesyłany do DAC.

Zmienna `DAC_DMA_sine12bit` przechowuje kolejne wartości, które przesyłane będą (cyklicznie) do modułu DAC (zgrubne przybliżenie sinusa o amplitudzie 2048 i offsecie 2048).

```
void MY_DAC_init(void)
{
	GPIO_InitTypeDef GPIO_InitStructure;

	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA , ENABLE);	// wyjscie DAC

	//inicjalizacja wyjścia DAC
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init(GPIOA, &GPIO_InitStructure);

	RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE); //DAC

	DAC_InitTypeDef DAC_InitStructure;

	//wyłączenie zewnętrznego wyzwalania
	//konwersja może być wyzwalana timerem, stanem wejścia itd. (szczegóły w
//dokumentacji)
	DAC_InitStructure.DAC_Trigger = DAC_Trigger_T6_TRGO;
	//wyłączamy generator predefiniowanych przebiegów wyjściowych (wartości
//zadajemy sami, za pomocą opowiedniej funkcji)
	DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
	//włączamy buforowanie sygnału wyjściowego
	DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
	DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude = DAC_LFSRUnmask_Bit0;

	DAC_Init(DAC_Channel_1, &DAC_InitStructure);

	DAC_SetChannel1Data(DAC_Align_12b_R, 0x000);

	DAC_Cmd(DAC_Channel_1, ENABLE);

}
```

Proszę zwrócić szczególną uwagę na ustawienie pola DAC_Trigger – układa DAC wyzwalany będzie za pomocą Timera numer 6.

Aby skonfigurować timer do wyzwalania DAC należy:

```
void MY_DAC_initTimerForUpdating(void)
{
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);

	TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
	TIM_TimeBaseStructure.TIM_Prescaler = 42-1;
	TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
```

```
    TIM_TimeBaseStructure.TIM_Period = 20;
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;
    TIM_TimeBaseInit(TIM6, &TIM_TimeBaseStructure);

    TIM_SelectOutputTrigger(TIM6, TIM_TRGOSource_Update);

    TIM_SetCounter(TIM6, 0);
    TIM_Cmd(TIM6, ENABLE);
}
```

Proszę zwrócić uwagę na `TIM_SelectOutputTrigger` które uruchamia generowanie sygnałów do wyzwalania DAC.

Zadanie:

Skopiować przedstawione fragmenty kodu, a następnie wywołać funkcje **`MY_DAC_init`, `MY_DAC_initTimerForUpdating` oraz `MY_DMA_initM2P`** i dodać pętlę główną (**nieskończoną**). Rezultatem działania programu powinien być głośny dźwięk generowany przez głośnik na płytce rozszerzeń. Spróbować zmodyfikować częstotliwość generowanego dźwięku (zmienić ustawienia timera).

2. Zadania do samodzielnego wykonania realizacji.

- Bazując na przykładach dotyczących transferów P2M oraz M2P napisać funkcję, która konfiguruje połączenie DMA do przesyłania danych z pamięci do pamięci (**sprzętowe memcpy!**).

  Wskazówki:

  - tylko DMA2 umożliwia transfer typu memory to memory,
  - na początek przesyłać tylko jeden bajt, gdy to zostanie osiągnięte przesyłać tablicę o określonej długości,
  - w celu odczekania aż transfer zostanie zakończony wykorzystać funkcję (**przykład dla DMA2 Stream0**):

```
void waitUntilDMATransferHasFinished(void)
{
	while(DMA_GetITStatus(DMA2_Stream0, DMA_IT_TCIF0) ==
SET)
	{
		asm("nop");
	}
}
```

- Przeskalować wartość odczytywaną z przetwornika ADC na wartość napięcia wyrażoną w woltach.
- Zmieniać jasność poszczególnych kolorów diody RGB (PB5, PC6) w zależności od wartości odczytanej z ADC. Wykorzystać PWM (TIM3, kanał 1 i 2).

- Wykorzystać wartość odczytaną z przetwornika ADC do zmiany częstotliwości dźwięku.